Anechoic chamber characterization for trustful evaluation of wireless protocols

Yassir M'rabet

August 31, 2017
Version: Final Report

Nice Sophia University

Inria

Diana team

Anechoic chamber characterization for trustful evaluation of wireless protocols

Yassir M'rabet

Reviewer Frederic Precisoso

Nice Sophia Antipolis University

Supervisors Walid Dabbous

Thierry Turletti

Mohamed Naoufal Mahfoudi

Diana team, Inria

August 31, 2017

Yassir M'rabet

Anechoic chamber characterization for trustful evaluation of wireless protocols

August 31, 2017

Nice Sophia University
Diana team
Inria
2004 Route des Lucioles
06902 Sophia Antipolis

Abstract

With the constant evolution of Wireless networking, it becomes difficult to monitor the performance of protocols in realistic settings because we encounter non controllable factors present in the wireless environment that impact the evaluation experience outcome. The motivation behind this work is to be able to trustfully evaluate wireless protocols in a given wireless environment because the evaluation is environment dependent. Anechoic chambers offer a controlled environment where a fair assessment of these network protocols is possible due to the absence of non controllable factors. They allow us to identify parameters of importance of our environment and moreover we can inject in a controllable way interference and assess their effect on the propagation. Unlike other classical anechoic chambers R2lab offers enhanced propagation realism due to reflexions inside the room causing multi-path transmission between a sender and a receiver. Our work thus aims at characterizing the anechoic chamber R2lab and derive fingerprints of this wireless environment to make it a candidate for on coming protocol evaluation. We first determine the factors of influence on the received signal strength metric. Then we fit the transmission inside the lab with the existing theoretical models for received signal strength prediction. Knowing that the Friis[1] transmission equation is the representative model of classic anechoic chambers, we determine how much it is representative for this anechoic chamber. Finally we intend to recognize line of sight communications from non line of sight ones between couple of nodes as a confirmation of the characterization of R2lab with the factors of influence found. We achieve a success rate of 66% for LOS identification.

[1] https://fr.wikipedia.org/wiki/%C3%89quation_des_t%C3%A9l%C3%A9communications

Acknowledgement

This research was closely supervised by my tutors Mr. Dabbous and Mr. Turletti that I thank for their time and support. I also want to thank my friend and tutor Mr. Mahfoudi for his assistance and for his comments that greatly improved the manuscript.

Contents

1 Introduction **1**

1.1 Motivation for trustful evaluation of wireless protocols 1

1.2 Context and motivation for choosing R2lab as a wireless environment 1

1.2.1 Presentation of R2lab 2

1.3 Challenges: Impact of multi-path clusters 3

1.4 Related work 3

2 R2lab characterization Methodology **5**

2.1 Factor analysis and fingerprinting 5

2.2 Experience orchestration 6

3 Factor analysis and Model fitting **9**

3.1 Exploring the factor space 9

3.1.1 Controllable factors 9

3.1.2 Non controllable factors 13

3.2 Model fitting of R2lab 16

3.3 Recapitulation of factor analysis results 18

4 Line of sight and non line of sight identification **21**

4.0.1 Steps of computation 24

4.0.2 The experiment 25

4.0.3 Testing part 26

4.0.4 CSI instead of RSSI 28

4.1 Recapitulation of LOS identification results 32

5 Conclusion **33**

5.1 Future Work 33

Bibliography **35**

Appendices **37**

A Code to extract RSSI **39**

A.1 Python3 script to automate the experiment 39

A.2 Bash script to interact with wireless card 50

A.3 Python3 script for post processing 54

1 Introduction

> *Research is to see what everybody else has seen, and to think what nobody else has thought.*
>
> **— Albert Szent-Gyorgyi**

1.1 Motivation for trustful evaluation of wireless protocols

Wireless protocols are a set of defined standards and policies comprised of rules, procedures and formats that define communication between two or more devices over a network. The OSI stack is a conceptual model made of layers of protocols. While there is always room for improvement of the underlying protocols, monitoring their performance becomes particularly hard in realistic settings. Especially, in the wireless domain where signal's propagation behavior is environment dependent, hardly controllable, and widely variable. The Inria R2lab chamber [4] is both equipped with RF absorbers for reducing signal propagation phenomena as electromagnetic reflections, scattering and a Faraday cage for blocking external radio wave interference. A controlled environment offers the choice of artificially inducing signal propagation phenomena, which helps for assessing the network protocols in different types of environments without suffering from the randomness of real life wireless testbeds. Knowing the factors of influence in a given environment will give guidelines for a trustful evaluation inside that environment. While wireless equipments communicate through the physical medium using these wireless protocols, before evaluating the protocols we need to characterize the wireless environment in use to be sure that the evaluation result is not biased by the environment's response.

1.2 Context and motivation for choosing R2lab as a wireless environment

Among the wireless environments that are possible to characterize, each one brings a set of pros and cons. For instance an outdoor wireless environment will be

realistic but not controllable while an emulation software will be controllable but will lack physical medium realism. A distinction is made here between the protocol medium realism which comprises the communication protocols and the physical medium realism which is only relevant to mention in real transmissions. The figure 1.1 summarizes the pros and cons for each environment. Anechoic chambers are

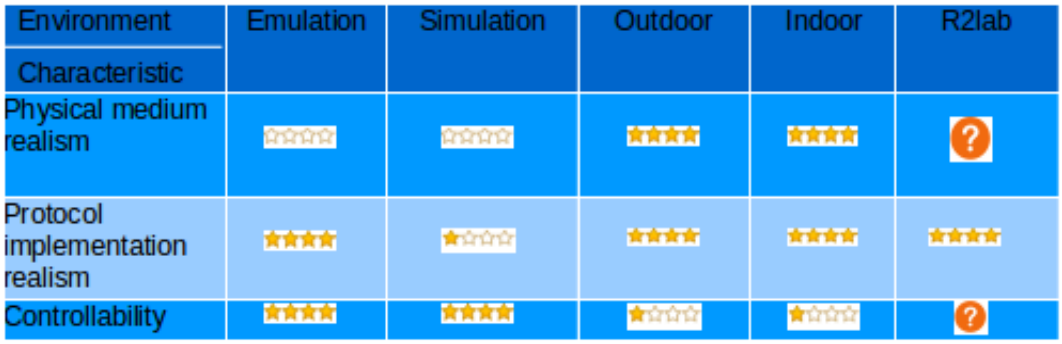


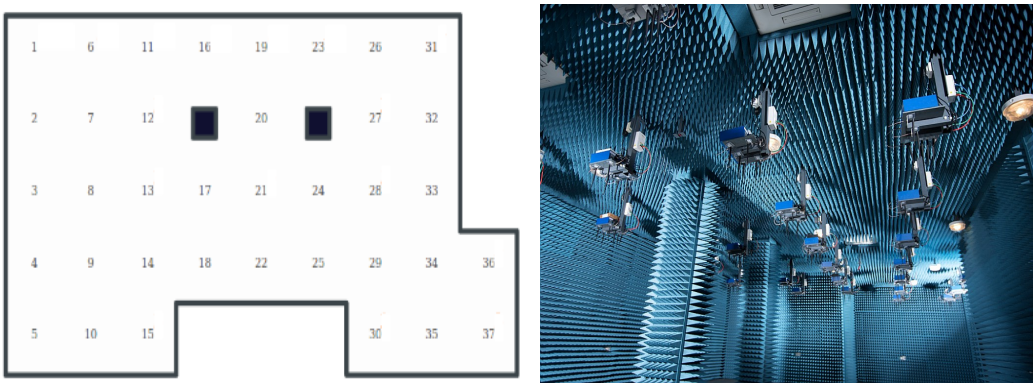
Fig. 1.1.: pros and cons of wireless environment

controllable but have poor physical medium realism due to the influence over the waves caused by removing all the propagation phenomenons (Reflexions, diffractions, scattering etc) which makes them a useful tool for evaluating wireless protocols, but how representative of the reality are they?
R2lab on the other hand is not like other anechoic chambers, we denote the presence of metallic caches surrounding the nodes that cause reflexions inside the room which makes it a good candidate for network experiments due to its increased realism. The object of our study "R2lab" has the specificities of an anechoic chamber while benefiting from the realism brought by the propagation phenomenons. But from this fact arises the need to characterize the transmission inside of R2lab. Moreover the wireless evaluation should be reproducible and trustful. The wireless environment chosen present the advantage of being located locally in Sophia Antipolis with a possible ssh connection for reproducibility of results.
On an evaluation test, we pick a metric and compare the metric change to conclude if a protocol has been enhanced. Thus in this work we will choose a metric and assess the impact of factors on that metric. Our motivation therefore is knowing a priori the outcome of a metric on a two nodes communication experiment. We want to define every couple of nodes interaction with respect to influent factors and their effect then bring out a representative model for the transmission inside R2lab.

1.2.1 Presentation of R2lab

R2lab is an open tested located in an anechoic chamber for reproducible research in wireless WiFi and 4G/5G networks. The R2lab platform sits in an insulated anechoic chamber of 90m2. It hosts thirty-seven nodes scattered on a fixed grid. A view of R2lab is shown in figure 1.2. The black squares in 1.2(a) are pillars, we can see

them in the figure 1.2(b). They constitute obstacles to the signal propagation. The numbers in 1.2(a) are the nodes identifiers.



(a) R2lab seen from above **(b)** R2lab inside view

Fig. 1.2.: View of R2lab

1.3 Challenges: Impact of multi-path clusters

The waves propagating inside R2lab are subject to reflexions and diffractions on the metallic caches of nodes which causes a multi-path transmission of waves between two nodes. We end up having, at the level of the receiver, a sum of constructive and destructive interference that introduces uncertainty on the signal's behavior. Moreover the signal can be affected by the correlation of some factors, for example the multi-path transmission and the frequency. The frequency in fact corresponds to a given wavelength. Different wavelengths will take different paths when reflecting and will consequently give different radiance diagrams in the room.
The nodes characteristics constitutes the second obstacle in our work, in fact each node has two wireless cards : Intel 5300 and Atheros 93xx and each constructor may implement different algorithms for a better signal reception. We also denote that each wireless card has three antennas it can use for transmission. Some wireless cards offer the possibility to set a mask for antennas, the mask then controls how many antennas will be used for the communication. Moreover the state of these equipments is to be taken in consideration because the performance is impacted when nodes heat to a certain level.

1.4 Related work

- R2lab has been used as a testbed to extend Mininet emulation platform from wired to wireless [1]. In fact the experiment results in R2lab were compared to the Mininet-WiFi propagation models for validation. Furthermore, the work attempted to proof the ability to replay network conditions from a real testbed

to reproduce in the emulated environment the expected behavior from a real world experiment. My work is understanding the behavior of propagation inside R2lab while this work [1] is by default taking R2lab as a representative realistic propagation environment. While realistic environments include a great amount of randomness, the models validated in the emulation software should be used in a likely similar environment conditions. In fact characterizing an environment is a step before using the environment as a testbed.

- While wireless experimentation is complex to set up and run, the paper [2] strove to five guidelines to avoid pitfalls that can occur during experimentation and thus present a methodology to conduct experiments in wireless networks. The methodology, if followed properly, facilitates the reproducibility of experimentation results and increases the accuracy of measurements performed. This work has given leads about where to look when anomalies are detected. While this work has been conducted in a different testbed than the one currently studied, we investigated possible pitfalls previously detected applied to the case of our wireless environment. In fact the goal of [2] is quite similar to ours in the sense that we also provide guidelines for running experiments projected in R2lab.

- In a study about LOS identification [3] the PHY layer information is explored to identify LOS dominant conditions with a commodity WIFI infrastructure. This work was envisioned as an early step towards a generic pervasive, and fine-grained channel profiling framework, which paves the way for WLAN based communication, sensing and control services in complex indoor environment. LOS identification is a part of the characterization of an environment. Having a way to perform LOS identification is useful in the sense that the outcome of the experience can be compared to the ground truth to confirm the factors of influence of our environment. We thus intend to provide a candidate metric based on [3] that allows LOS identification in our environment.

- Classical anechoic chambers perfectly absorb electromagnetic waves incident on other than the receiver making the distance the most influent factor on the signal strength. In this case the Friis path loss model equation is the model representing the transmission inside anechoic rooms as shown in [5]. It represents the free space signal strength propagation. While R2lab is not like other anechoic chambers we will answer the question of the representative model in our environment case and determine how close the model is to the real transmission.

2 R2lab characterization Methodology

We recall that we want to characterize the transmission inside R2lab. We want to know a priori the conditions of reception between every couple of nodes inside R2lab. To do that we intend to find the factors of influence in a two nodes transmission. In order to find the factors of influence we use a factor analysis methodology.

2.1 Factor analysis and fingerprinting

The approach considered in this work is an experimental one that considers the object of study “R2lab” as a black box with some inputs, we call them factors, that have an impact on a given metric. The role of the metric is to relate on the goodness of a communication and give a specific expected result of the metric for a given configuration of factors. By comparing the outcome metric of two experiments we can make a qualitative evaluation between the two experiments. Examples of metrics are the received signal strength (RSSI), the channel state information (CSI), the packet loss etc.

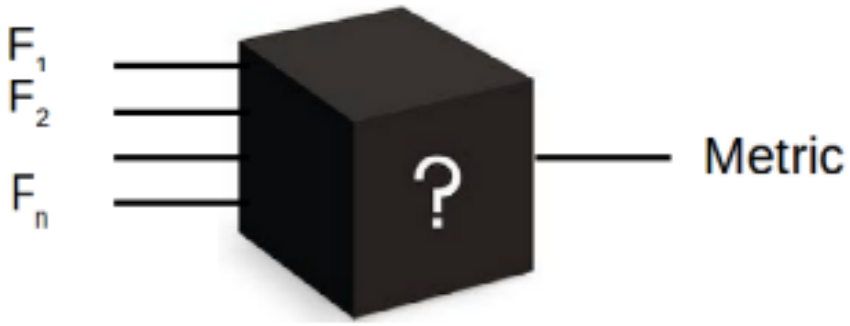


Fig. 2.1.: Experiment design

In our case we choose the RSSI as a metric at first. It is the amplitude of the received signal at the level of the receiver in dBm (i.e. milliwatt in logarithmic scale). The other metric possible to use is the CSI. It represents the power strength of each sub carrier when a packet is split in chunks of data. The CSI gives more detailed information about the reception. We non less decide to choose the RSSI because it is

a widely used metric and fairly accessible that eases reproducibility in commodity hardware. The factors that have an impact on the metric can be one of the following categories:

a. Controllable factors: the factors that we can change at the level of each node, like frequency and transmission power.

b. Non controllable factors: the factors that cannot be changed, like the geographic position of the nodes inside the room. We also can imagine that each wireless card constructor introduces a part of uncertainty due to different software implementation. The state of the equipments can also be classified in this category because it impacts performance if they heat over a certain level.

Our methodology is the following:

1. Exploring the factor space: we affect value levels for factors and assess the effect of their change on the RSSI.

2. Model fitting: We represent the transmission inside R2lab along with theoretical models that predict the RSSI to identify which model best fits the lab.

3. Line Of Sight and Non Line Of Sight identification: We intend to identify LOS communication from NLOS ones as a confirmation of previously found factors. The result of the LOS identification compared with the ground truth will proof the possible presence of other factors. This chapter is a confirmation of the characterization of R2lab with the previously found factors.

Our finding will forge fingerprints of our environment and allow us to construct radio maps of R2lab.

2.2 Experience orchestration

The deployment environment has been chosen to be simple to implement in the perspective of having reproducible experiments. Indeed we wrote python scripts running on commodity hardware along with bash scripts to interact with the wireless interfaces of nodes. The Nepi-ng tool was used for running and orchestrating network experiments on hosts composed of two existing libraries implemented in python3:

- Apssh[1]: a library to create ssh connections and instantiate ssh jobs, which are commands that will be run on the remote nodes.

- Acynciojobs[2]: a library to run asynchronously the ssh jobs on the nodes.

The experiment consists on generating some traffic between nodes and monitoring the RSSI at the level of the node receiver, the step by step process is the following:

1. Create ssh connections using the Apssh library.

2. Run a bash script on each node to create an ad-hoc network at a given frequency.

3. Run tcpdump on each node to monitor the on coming traffic which will be written in packet capture files.

4. Generate traffic between nodes.

5. Filter the column of interest in the packet capture files corresponding to the RSSI of the communication.

The full code for orchestrating a traffic experiment between nodes and monitoring the RSSI was initially designed by the Diana team, then I modified the design to make the run simultaneous making the run faster and optimizing the space the outcome was taking. I also added some modification to the script to infer more metrics. The modified code can be found in the annex A.
At this point we possess values of RSSI for each two nodes communication. Based on these values I generate RSSI heatmaps (static and interactive) of R2lab using the open source plotly library. Moreover I construct a database of RSSI values to monitor the changes on the same nodes interaction over time. After that I fit the histogram of RSSI values and perform a goodness of fit test. I also provide a graphical way to upload RSSI files (outcome of the RSSI generation script) and plot RSSI heatmaps as well as daily changes of RSSI and fit of the lab with the django [3] framework which is a high-level Python Web framework. Due to the size of codes, I provide them in github[4] instead.

[1]https://pypi.python.org/pypi/apssh/0.5.6
[2]https://pypi.python.org/pypi/asynciojobs/0.5.4
[3]https://www.djangoproject.com/
[4]All codes used in this work can be accessed in https://github.com/YassirMr/Internship

3 Factor analysis and Model fitting

3.1 Exploring the factor space

3.1.1 Controllable factors

To find the factors of influence on the RSSI we select a set of controllable factors then assign for each factor a low and high level and generate traffic on the levels chosen. We then monitor the RSSI for each experiment and describe the effect of factors on the metric. The factors and levels chosen are the following:

- Transmission power: 5dBm and 14dBm.
- Physical rate: 6Mb/s and 54Mb/s.
- Distance: 1.36m (corresponding to the node sender 4 and receiver 9 distance) 10.88m (corresponding to the node sender 4 and receiver 36 distance).
- Frequency: 2412MHz and 5180MHz.

In the experiment node 4 will be the sender and node 9, 36 receivers (figure 3.1). Each time we will change one and only one factor.

Fig. 3.1.: Nodes evolved in the experiment

The result of runs for the experiment are given below:

Transmission power (dBm)	Physical rate (Mb/s)	Distance (m)	Frequency (MHz)	RSSI (dBm)
14	6	1.36	2412	-33.64
14	6	1.36	5180	-30.54
14	6	10.88	2412	-58.63
14	6	10.88	5180	-63.88
14	54	1.36	2412	-34.52
14	54	1.36	5180	-29.82
14	54	10.88	2412	-59.16
14	54	10.88	5180	-63.45
5	6	1.36	2412	-42.33
5	6	1.36	5180	-39.95
5	6	10.88	2412	-67.09
5	6	10.88	5180	-73.17
5	54	1.36	2412	-42.30
5	54	1.36	5180	-38.23
5	54	10.88	2412	-67.06
5	54	10.88	5180	-71.42

We present the factors of influence on the RSSI with an interaction diagram:

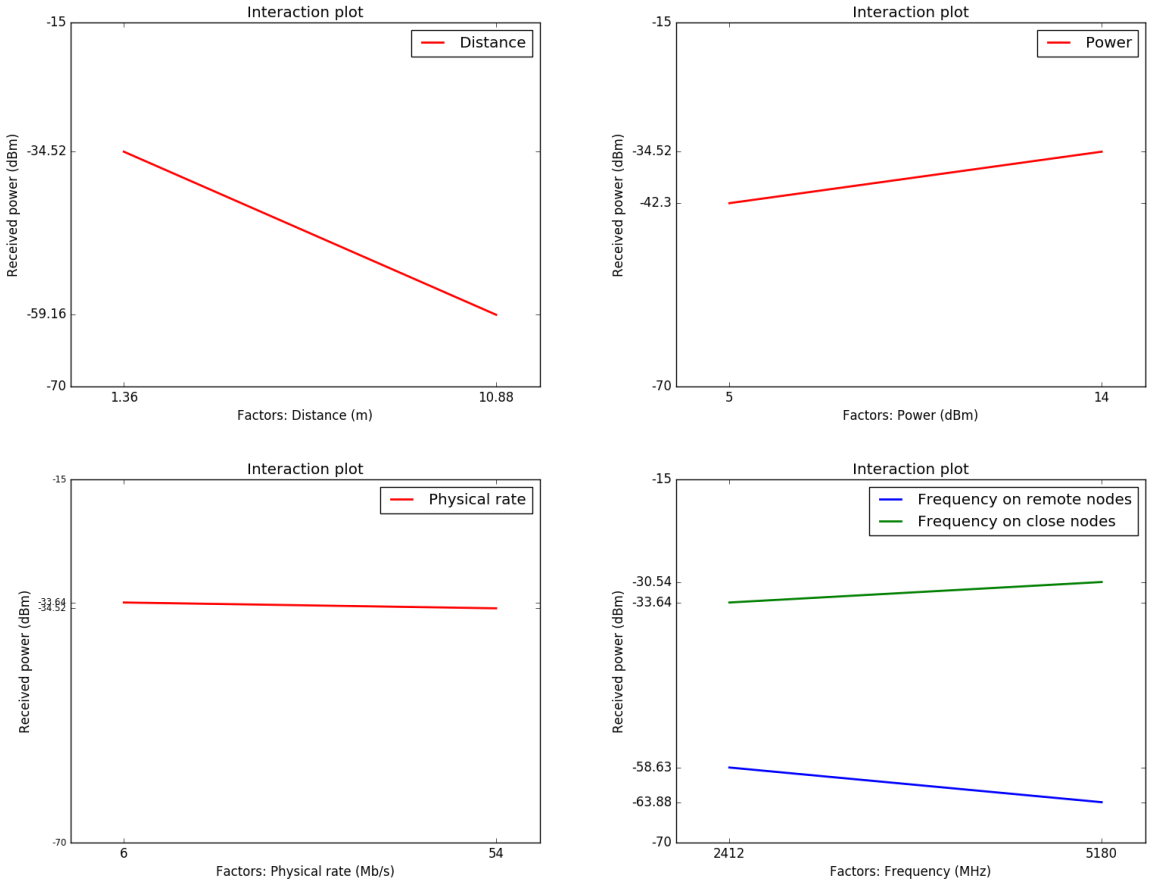


Fig. 3.2.: Interaction plot of Factors

While the transmission power and distance factors clearly impact the RSSI in one way and the physical rate does not show a consequent impact, the frequency effect on the other hand is not straightforward. We notice on remote nodes the 2GHz band achieves higher RSSI and the opposite on closer nodes. To understand the impact of the frequency isolate the frequency factor and make a RSSI heatmap of the room in 2GHz and 5GHz. A heatmap in this case will consist of a node sender and all the other nodes receivers, each node will display the RSSI got from the sender. In theory the transmission in 5GHz is more prone to signal degradation with distance, thus we would expect a warmer heatmap in the 2GHz band. While we want to characterize only the effect of the frequency on the RSSI, we pick a mask of 1 and use one antenna for emission and reception. We get the following heatmap when node 19 is the sender:

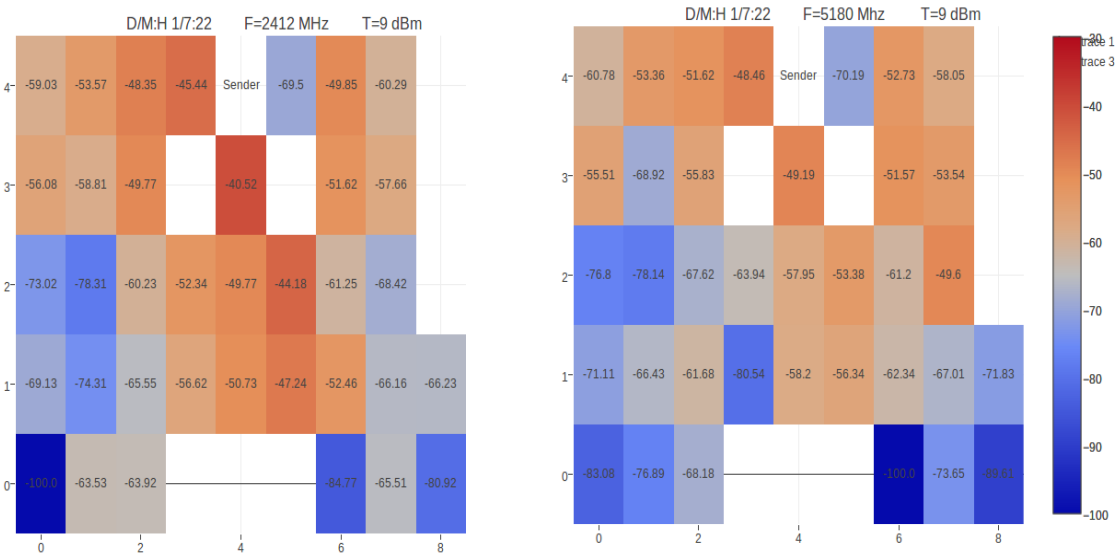


Fig. 3.3.: Heatmap of R2lab in 2GHz and 5GHz

First we notice values of -100dBm of RSSI, they correspond to the fact that the node did not receive a signal above sensitivity and thus could not decode the packet, it can happen when only one antenna is emitting. Besides we notice a mostly warmer heatmap in 2GHz but we remark the presence of outliers, for example node 9 and 32. The behavior or the outliers is explained by the fact that the scattered waves on pillars and the reflexions on the metallic caches of other nodes cause multi-path transmissions, and depending on the wavelength at the given frequency it is causing a constructing interference. Indeed we denote the frequency to wavelength matching:

- 2412 MHz : 12.43 cm.
- 5180 MHz : 5.8 cm.

We conclude that the most effective factors are, in a descending order:

- Distance
- Transmission power
- Frequency if there is no multi-path transmission.

Although the 2GHz band covers more surface some applications may need to switch into 5GHz, in fact higher frequencies have more bandwidth and therefore can carry more information, yet there is a condition for higher rates. The SNR (i.e. the noise subtracted from the RSSI) should be higher than a value in order to access some physical rates as shown in figure 3.4. We recall that the noise is a parasite signal that disturbs the communication. It is inherent to the card, impacted by the temperature of equipments, the state of antennas and the band used, indeed each frequency band has a corresponding value for noise. While the RSSI in the two heatmaps of figure 3.3 is fairly the same we need to compare the noise in the two bands to know which band is more adequate for higher rate. To do so, using the same logic of the script seen in the methodology, we extract the noise from the nodes right before we generate traffic, like that we have the noise state of the room for a given node sender. We present a heatmap of noise in the two bands, each cell is containing the power of noise in dBm (figure 3.5).

Rate(Mb/s)	1	2	5.5	11	6	9	12	18	24	36	48	54
SNR(dB)	4	6	8	10	4	5	7	9	12	16	20	21

Fig. 3.4.: SNR needed to access physical rates[1]

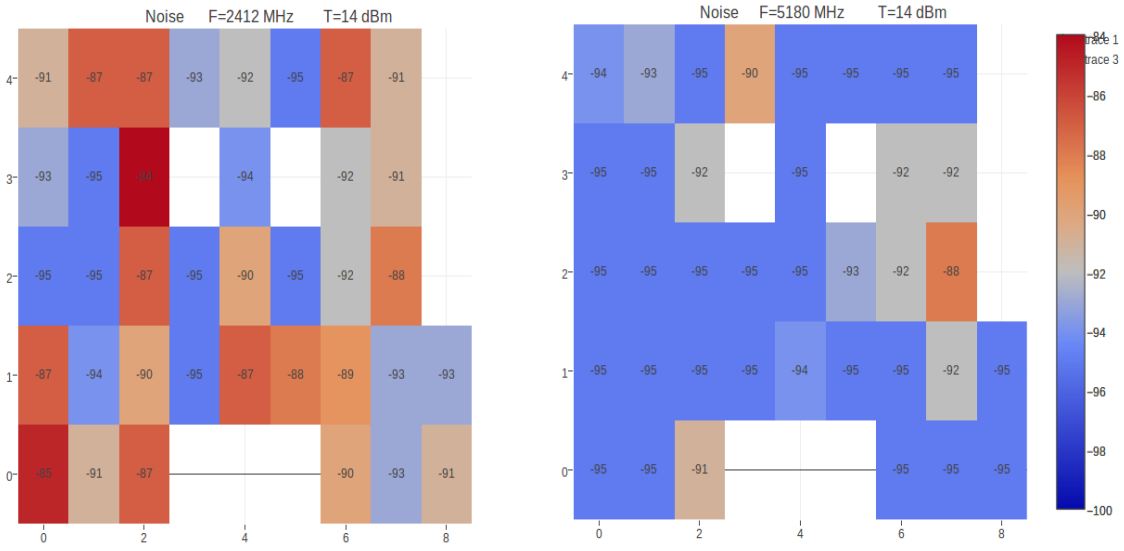


Fig. 3.5.: Heatmap of noise in 2GHz and 5GHz

We notice that the noise is generally lower in 5GHz, therefore the 5GHz band is more suitable for higher rates due to the little noise.

3.1.2 Non controllable factors

The geographic position of nodes inside the room is a non controllable factor. The experiment in this case aims to find if this factor impacts the RSSI received by each antenna. We select the nodes positions that we will assess and then present the received signal strength within each antenna for each experiment as a histogram. The direct neighbor nodes can be in one of the following positions:

(a) Row neighbors **(b)** Column neighbors

Fig. 3.6.: Possible relative antenna positions for direct neighbors

We consider node 21 as a sender and the adjacent ones as receivers, the nodes involved in the experiment are:

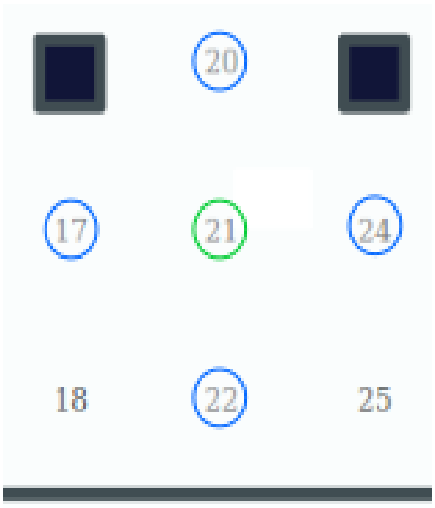
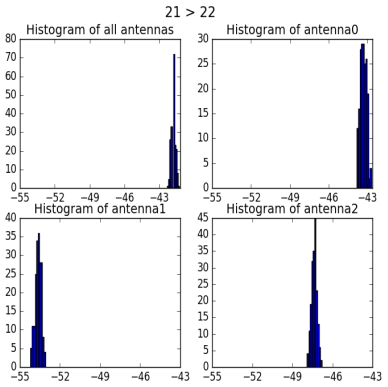
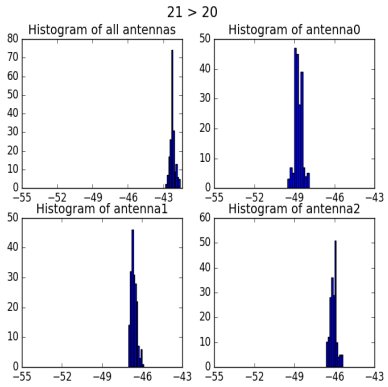


Fig. 3.7.: Nodes involved

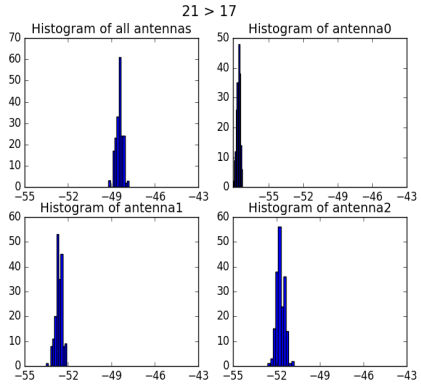
We present the histograms of RSSI for each antenna:



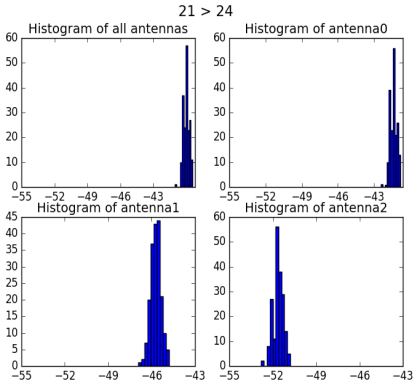
(a) Bottom neighbor



(b) Top neighbor



(c) Left neighbor

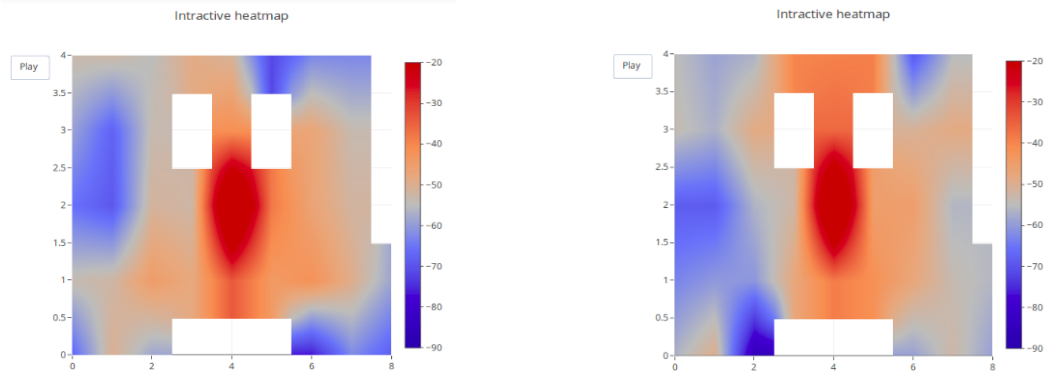


(d) Right neighbor

Fig. 3.8.: Histograms of received signal strength for each antenna

In addition to the three values of the signal strength at each receiver antenna, we have access to a fourth value of RSSI. It is the aggregation of signal strength by all antennas. The technique used to infer this value is the Maximal Ratio Combining (MRC). It consists on extracting the maximum energy of bits at each antenna and aggregate the best symbol energies to get the highest value for the received signal strength. The first remark is that the RSSI value of MRC is always better than the values for other antennas. Then we notice that in the figure 3.8(a) the mean of antenna 0 is higher than the one of antenna 2 and inversely in the figure 3.8(b). This is due to the position of the receiver antennas, each time the antenna receiving better is the front one not hidden by the other antennas. The impact of having an obstacle as small as another antenna makes a difference in the RSSI. For the experiment involving the same row nodes (3.8(c) and 3.8(d)), in that configuration no antenna is hidden by a given obstacle and yet we still notice that each time we have a different antenna that is receiving better. This might be due to the radiance diagram of antennas, when the antennas are radiating, each time a different antenna is in a better receiving zone of radiance, to show this radiance zone, we worked on making interactive heatmaps that show how each antenna receives power, we

present a figure of node 21 sender and change the receiving antennas.



(a) Antenna 0 receiving **(b)** Antenna 2 receiving

Fig. 3.9.: Interactive heatmaps

The geographic position of nodes impacts which antenna is receiving better, thus we can choose a specific class of antenna positions in order to evaluate an experiment and remove changes caused by this factor.
Looking at the various wireless card constructors that exist we can imagine that each constructor is trying to improve the received signal strength for a better quality of experience, so are there other non controllable factors arising from constructors themselves ?
To answer this question, let us first show a typical RSSI monitoring over time between two nodes:

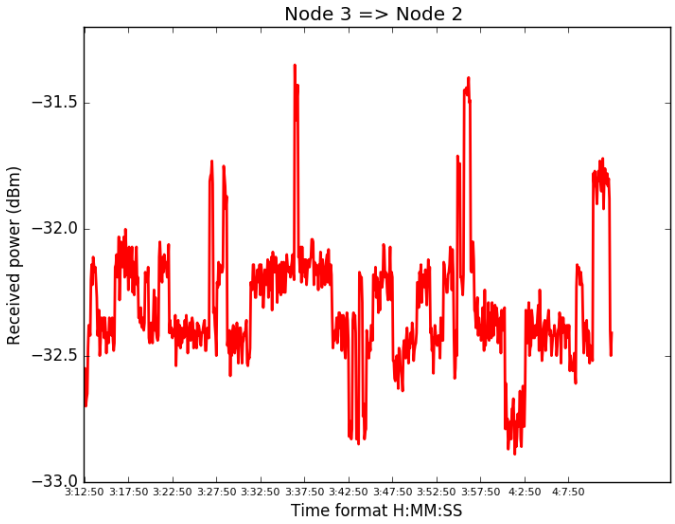
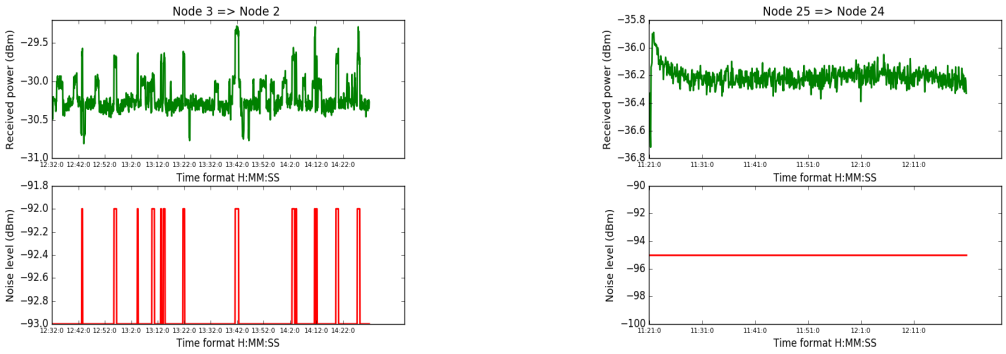


Fig. 3.10.: Received signal strength over time

We notice a fluctuation of around 1dB of the signal strength. As notified in [2] some wireless cards include a periodical recalibration process that can cause an alteration

of the signal received power. To find out if this RSSI fluctuation is impacted by a change in the noise we re-run the experiment below between the same nodes and monitor both the RSSI and noise over time (3.11a) and seek a communication with a constant noise over time (3.11b). The constant noise communication was found by running the same experiment multiple times.



(a) Non constant noise case **(b)** Constant noise case

Fig. 3.11.: Received signal strength over time

We notice that whenever the noise increases, it is sensed by the wireless card and this causes the RSSI to increase in response while in a constant noise situation, we do not notice a consequent fluctuation in the RSSI. It means that whenever the RSSI is monitored and some fluctuations are noticed, before making a conclusion, the noise should be monitored in the adequate band used. Moreover, removing the effect of noise is a step to make the output usable for evaluation.

3.2 Model fitting of R2lab

Theoretical models are used to predict values of received signal strength based on factors, for instance if the transmission is indoor or outdoor, depending on the frequency of the transmission, on the distance between the two equipments, thickness and material of the obstacles etc. Our purpose here is to evaluate the theoretical models with a real transmission inside R2lab. The models chosen, due to their relevance, are the following:

1. Friis Telecoms equation [2] that represents the free space propagation [5]:

$$P_r = P_t + G_t + G_r + 20log_{10}\frac{\lambda}{4\pi R}$$

Where:

[2] https://fr.wikipedia.org/wiki/%C3%89quation_des_t%C3%A9l%C3%A9communications

- P_r/P_t : The power at the level of the receiver/sender antenna in dBm.
- G_r/G_t: The gain at the receiver/sender.
- R : The distance between the antennas in meters.
- λ : The wavelength at the corresponding frequency in meters.

2. The two ray ground reflected model [3] when the transmission between nodes is a result of an incident and reflected wave [6]:

$$P_r = P_t + G_t + G_r + 10log_{10}(G{h_t}^2{h_r}^2) - 40log_{10}(d)$$

Where:

- G : The gain at the level of the receiver.
- h_t/h_r : The height of the emitting/receiving antenna.
- d : The distance between the antennas.

3. The log distance path loss model[4] that represents a more generalized form of the path loss model can be constructed by modifying the free-space path loss with a path loss exponent that varies with the environments [5]:

$$P_T - P_R = PL_0 + 10\gamma log_{10}\frac{d}{d_0} + X_g$$

Where:

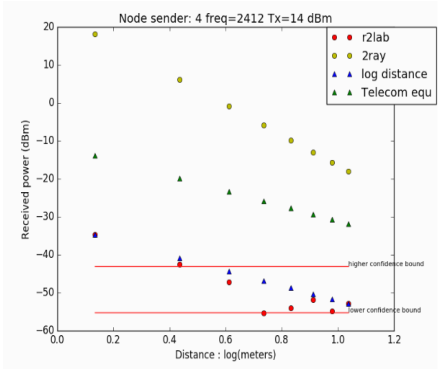
- P_R/P_T : the power at the level of the receiver/sender antenna in dBm.
- d : the length of the path.
- d_0 : the reference distance.
- PL_0 : the path loss at the reference distance d_0.
- γ : the path loss exponent, for our case we found a value of 2 achieving a good approximation.

[3] https://en.wikipedia.org/wiki/Two-ray_ground-reflection_model
[4] https://en.wikipedia.org/wiki/Log-distance_path_loss_model

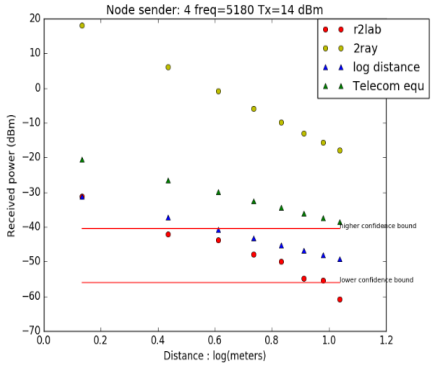
- X_g : the normal (or Gaussian) random variable with zero mean, reflecting the attenuation (in decibel) caused by flat fading. In case of no fading, this variable is 0.

The set of nodes involved in the experiment is the one of the longest row in R2lab (figure 3.12). Node 4 is considered the sender and the remaining nodes receivers. We monitor the received signal at each node and present the signal behavior with distance in 2GHz and 5GHz (figure 3.13).

Fig. 3.12.: Nodes involved in the experiment



(a) 2GHz band



(b) 5GHz band

Fig. 3.13.: Signal strength with distance along with models

Below the Mean square error for each fitting:

Mean square error	R2lab/Log-distance	R2lab/two-ray	R2lab/Telecom-equation
2GHZ	15.63	576.07	1996.53
5GHZ	41.11	277.07	1895.9

The experiment shows that the model fitting best the transmission inside the lab is the log distance model. The log distance model relates on the path loss of indoor communications where the Power decreases logarithmically with distance due to major attenuation. This is due to the fact that the reflexions inside R2lab introduces added attenuation of the RSSI along with distance.

3.3 Recapitulation of factor analysis results

- We have seen the factors of influence on the RSSI metric are the distance, the transmission power and the frequency.

- We also noticed that the geographic position of nodes play an impact on the RSSI received.
- We saw that antennas could be in different radiance zones.
- We detected the presence of a non controllable factor which is noise.
- We found the model fitting best the transmission inside R2lab is the log-distance path loss model.

Therefore the dominant effect on the RSSI inside R2lab is the distance coupled with the multi-path interference that introduces more signal degradation and makes the log-distance representative in our case.

4 Line of sight and non line of sight identification

The identification of line of sight communication (i.e. not blocked with an obstacle) will be compared to the ground truth to proof if the factors found are the only factors of influence. Our journey through constructing fingerprints of R2lab leads us to define if a communication is line of sight dominant or not when it happens. The state of the art of wireless communication stipulates that LOS signal strength follows the Rician distribution while NLOS signal strength follows Rayleigh distribution. Therefore we will fit the histogram of RSSI, detect which distribution is followed and examine if our environment follows the state of the art results.
To assess the fitting of a histogram we need to choose a goodness of fit test and a hypothesis statement along with a threshold value to reject or not the hypothesis. The test chosen is the Kolmogorov–Smirnov test (K–S test or KS test), it is a nonparametric test of the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution. Our

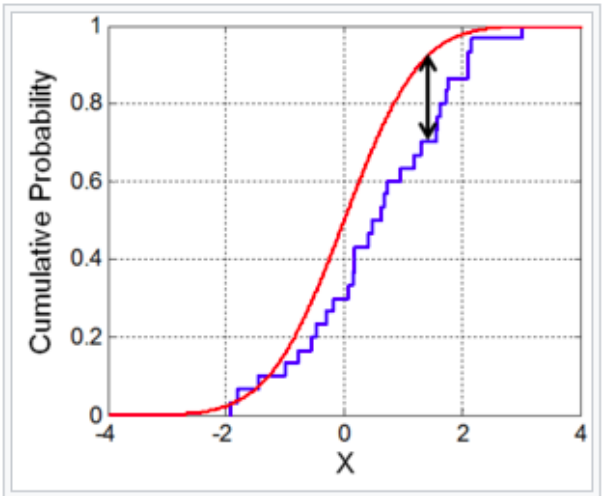


Fig. 4.1.: Illustration of the K-S test. Blue line is the ecdf of the data, red line is the fitting cdf, black arrow is the K-S statistic

hypothesis depend on the couple of nodes chosen, we first make the LOS experiment, for this case we stipulate the hypothesis:
H_{00} : The samples follow Rice distribution.
H_{01} : The samples do not follow Rice distribution.

Then we make the NLOS experiment and in that case the hypothesis is:
H_{10} : The samples follow Rayleigh distribution.
H_{11} : The samples do not follow Rayleigh distribution.

We select a p-value of 0.05 as a significance level. Note that when selecting a couple of nodes, each antenna could be following a specific distribution and thus the statistical test applies to each antenna.
The LOS experiment consists on taking two nodes where there is no obstacle between them and monitoring 2000 samples of RSSI. We present the histogram and the fitting done with Rice distribution as well as the mean value for each histogram, the standard deviation, the mean square error between an equally sized array generated by the distribution and finally the output of the K-S test.

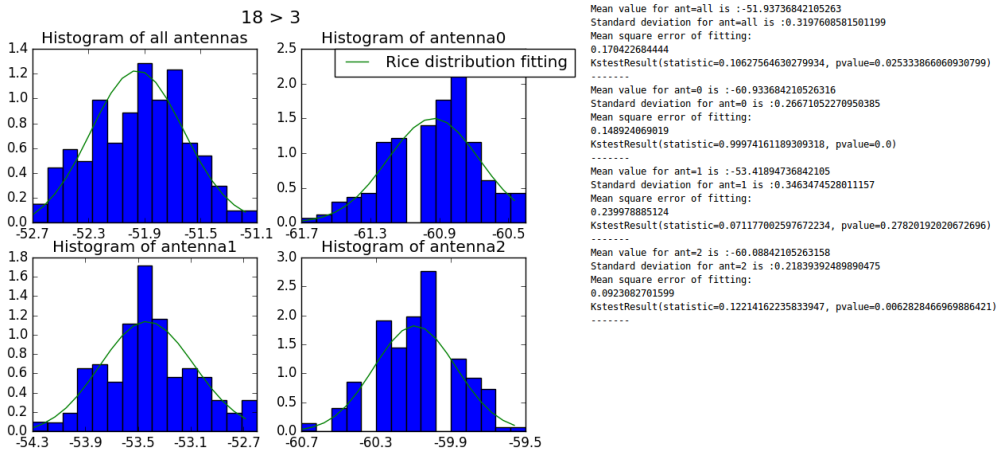


Fig. 4.2.: Rice fitting of received signal strength histogram in LOS case

We see that there is a strong evidence in favor of the H_{00} hypothesis for the antenna 1 but it is not the case for the antenna 0. In matter of fact the antenna 0 and 2 are in favor of the H_{01} hypothesis. We want to see if the antennas not following Rice distribution will follow Rayleigh distribution and be in the case of a NLOS case distribution. We attempt a Rayleigh fitting for the same communication.

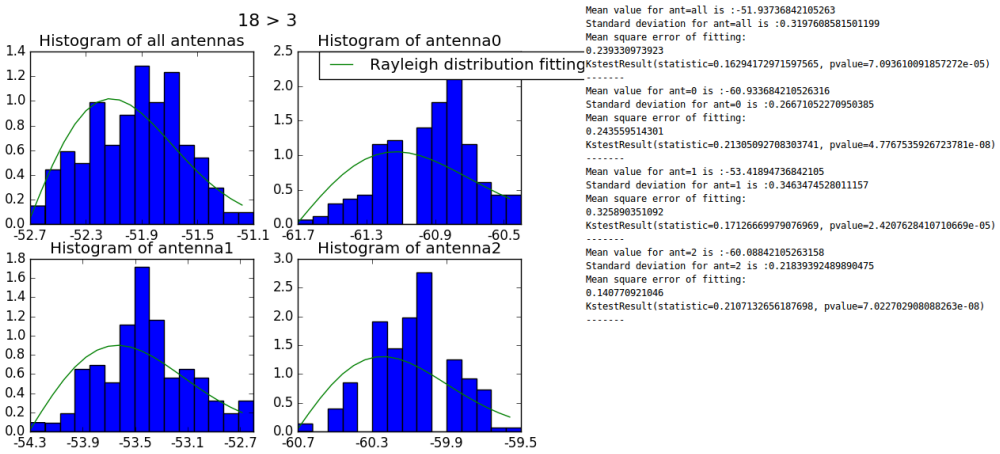


Fig. 4.3.: Rayleigh fitting of received signal strength histogram in LOS case

The result shows clearly that the test is in favor of hypothesis H_{11} . We are not in NLOS. Indeed discarding a hypothesis doesn't mean that the opposite is true. Let us now take two nodes separated with a clear physical obstacle and make to same experiment with a Rayleigh fitting for the NLOS case:

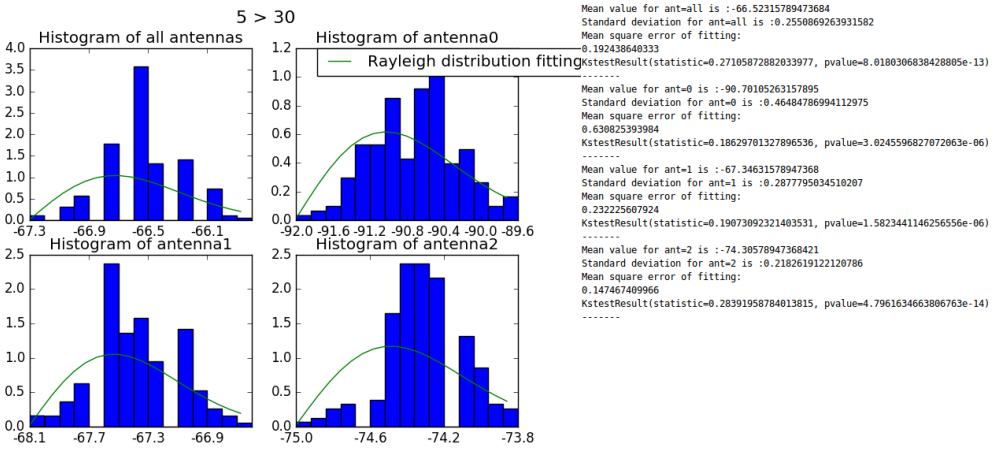


Fig. 4.4.: Rayleigh fitting of received signal strength histogram in NLOS case

The result clearly goes against H_{10} . The Rayleigh distribution doesn't fit our data, hence there is clearly an obstacle between nodes. The reason could be because there is not enough randomness that would be caused by moving nodes, in fact the Rayleigh fitting has been performed between a base station and a mobile client at the center of the densely-built Manhattan. The fact that nodes are static causes the RSSI to be more predictive and prevents us from concluding that a communication is NLOS using this technique. This is why we will try another technique [3] in our environment.

First we select the following hypothesis:

H0 : most of the power contained in the direct path.

H1: most of the power contained in the auxiliary paths.

The two candidates used to determine the nature of the communication in [3] were Rician-K factor and skewness and defined as follows:

1. **Rician-K factor** is defined as the ratio of the power in the LOS component to the power in the scattered NLOP paths. In this work we utilize a practical estimator for the Rician-K factor leveraging only the empirical moments [3].

$$\hat{K} = \frac{-2\hat{\mu_2^2} + \hat{\mu_4} - \hat{\mu_2}\sqrt{2\hat{\mu_2^2} - \hat{\mu_4}}}{\hat{\mu_2^2} - \hat{\mu_4}} \tag{4.1}$$

   Where $\hat{\mu_2}$ and $\hat{\mu_4}$ are the empirical second and forth moments of the measured data, respectively. A large K indicates strong LOS power and thus a high probability of LOS dominant conditions.

2. **Skewness** is a general metric relating on the skew shape of a distribution, the skewness relates on the asymmetry of the samples. Depending on the wireless environment, the LOS amplitude samples may present more or less symmetry than the NLOS samples. Mathematically it is defined as:

$$s = \frac{E\{x - \mu\}^3}{\sigma^3} \tag{4.2}$$

   Where x, μ and σ denote the measurement, mean, and standard deviation, respectively. A positive (negative) skewness indicates that the measured data spread out more to the right (left) of the sample mean.

4.0.1 Steps of computation

Given a set of N packets, we extract the RSSI and proceed to compute the K-factor and skewness as shown in equation (3.1) and (3.2). The LOS identification is formulated as a classical binary hypothesis test with LOS condition H_0 and NLOS condition H_1.
The hypothesis can be summered as:

$$\begin{cases} H_0 : K > K_{th} \\ H_1 : K < K_{th} \end{cases}$$

$$\begin{cases} H_0 : s < s_{th} \\ H_1 : s > s_{th} \end{cases}$$

Where K_{th} and s_{th} represent the corresponding identification threshold for the K-factor and skewness. The thresholds are computed according to measurements

conducted in R2lab for various host distances.

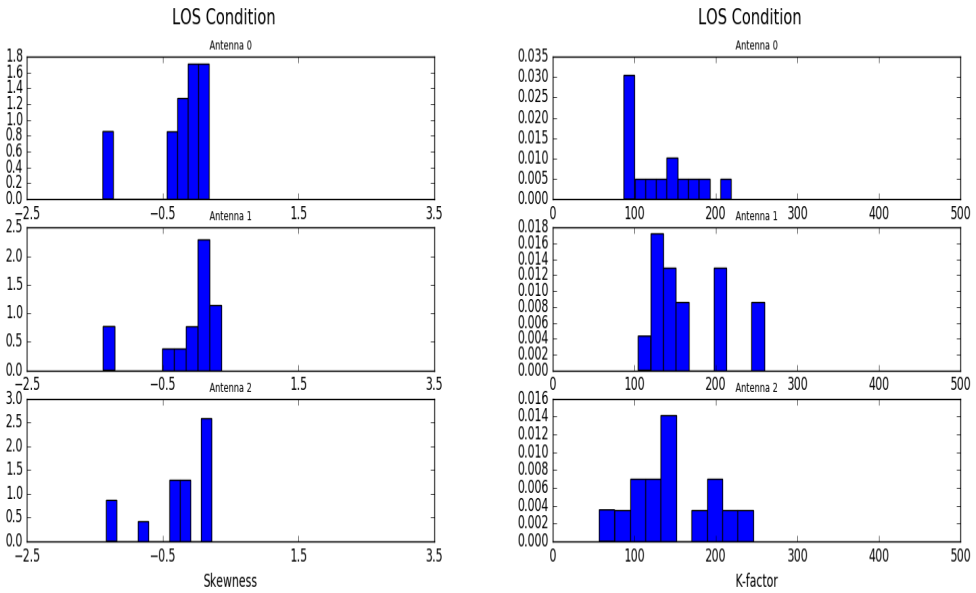
4.0.2 The experiment

We selected a set of nodes that are in LOS conditions and others in NLOS conditions then we computed the two metrics. We present their histogram and look for adequate thresholds of our hypothesis. The testing part will follow, it will test the thresholds with an other set of nodes that present LOS conditions and NLOS conditions. To ensure a test that the design did not corrupt we will use two groups of nodes, a design group and a test group.

Design	Test
1,2,3,4,5,6,7,8,9,12,13,14,17,18,20,21,22,24,25, 27,28,29,30,34	10,11,15,16,19,23,26,31,32,33,35,36,37

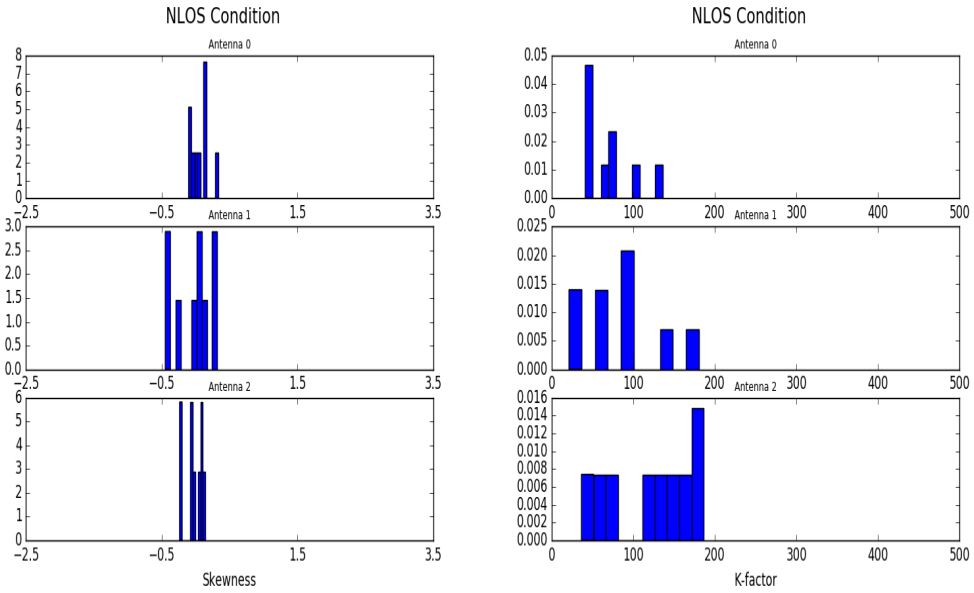
Fig. 4.5.: The design and test set of nodes

We present the histograms of the design interactions that are in line of sight then the ones in non line of sight. Then we compute the mean for the given interactions and define our threshold using these mean values.



Mean values	Antenna 0	Antenna 1	Antenna 2
Skewness	-0.2573	-0.147	-0.2712
K-factor	129.1	167.5	145.1

Fig. 4.6.: Histograms and mean values for Skewness and K-factor in LOS



Mean values	Antenna 0	Antenna 1	Antenna 2
Skewness	0.064	-0.04	-0.02
K-factor	70.5	84.2	121.5

Fig. 4.7.: Histograms and mean values for Skewness and K-factor in NLOS

We notice that in presence of a LOS (NLOS) communication the distribution is skewed to the left (right) and the K-factor is higher (lower), in our case the NLOS amplitude histogram is more symmetrical than the LOS one. The thresholds are then computed as the mean between the two samples means. Our experiment gives the values of threshold as following:

- -0.1 for the skewness and 99 for the K-factor of antenna 0.
- -0.1 for the skewness and 125 for the K-factor of antenna 1.
- -0.14 for the skewness and 133 for the K-factor of antenna 2.

4.0.3 Testing part

The testing experiment takes as input a set of RSSI values for a two nodes communication and outputs if we are in a LOS or NLOS condition by comparing the skewness and K-factor computed to the thresholds.
Sample output:

```
Rss_ant0:
Skewness
-6.554025987996666
LOS according to skewness
K factor
125.89240709762372
LOS according to k factor

Rss_ant1:
Skewness
-5.537459305829643
LOS according to skewness
K factor
92.29794474501296
NLOS according to k factor

Rss_ant2:
Skewness
-2.7910269759410915
LOS according to skewness
K factor
105.28797588736029
NLOS according to k factor
```

```
Rss_ant0:
Skewness
0.02252870692115199
NLOS according to skewness
K factor
159.88309741332694
LOS according to k factor

Rss_ant1:
Skewness
-0.09507269918612223
NLOS according to skewness
K factor
99.08917149143473
NLOS according to k factor

Rss_ant2:
Skewness
-0.042792986781874084
NLOS according to skewness
K factor
127.41393194763123
NLOS according to k factor
```

Fig. 4.8.: LOS and NLOS test

We see some false positive cases. The method will be showing a percentage of success that we want to determine. We thus take several LOS identification cases and NLOS identification ones from the test set defined earlier and use our thresholds to test the performance.
We find that the LOS identification case achieves:

- 63% of success for K-factor.
- 66% of success for the skewness.

While the NLOS identification case achieves:

- 66% of success for K-factor.
- 70% of success for the skewness.

We conclude that the skewness method for the identification is showing higher performance, but on how many packets is it valid ? The last step is to define how many samples are necessary to reach the previously found percentage of accuracy. We recall that we used 2000 samples to infer the thresholds and test them. But such a high number would be hard to implement, we reduce then the samples and test how

many samples are necessary to get a good estimation. We compute the percentage after changing the number of samples. The experiment result is presented below:

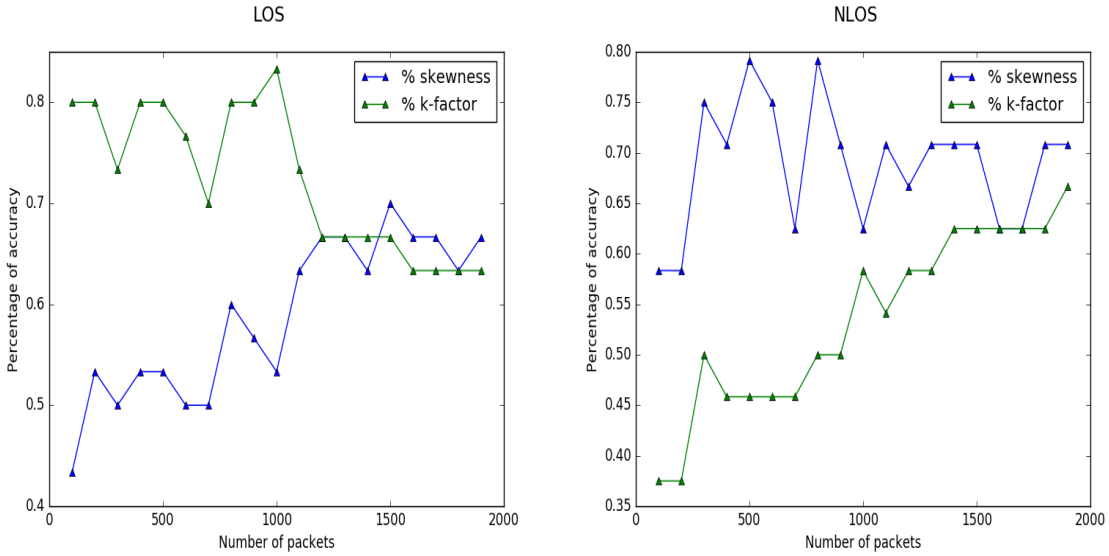


Fig. 4.9.: Impact of number of packets on the success rate

We find that 800 samples is a good compromise considering a relatively small number of packets and a fair percentage of success, in our experiment, it takes 4 seconds to monitor the RSSI of 800 packets.

4.0.4 CSI instead of RSSI

We expand the LOS identification technique to use the CSI metric instead of the RSSI, the CSI (channel state information) is fairly more accurate than the RSSI metric. When each packet is sent, it is decomposed of small chunks of data and each chunk is carried at a frequency. The CSI is the amplitude value for each sub-carrier. We will thus use the values of amplitude given by the CSI and find thresholds of skewness and K-factor for LOS communication and NLOS ones then we will also compute the percentage of success for other communications in the test set. Following the same logic we choose nodes between the design set and present the histogram of CSI amplitudes:

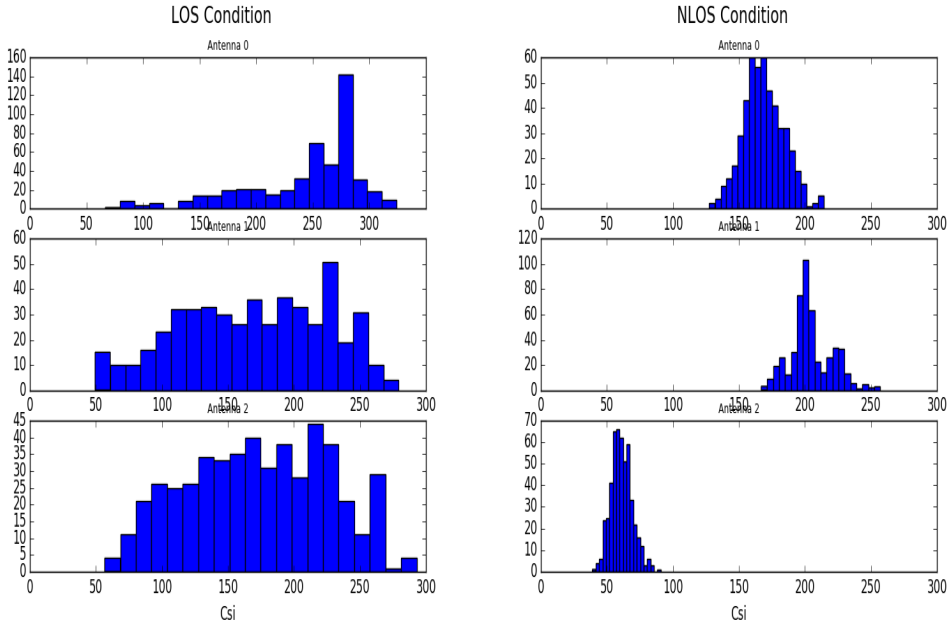
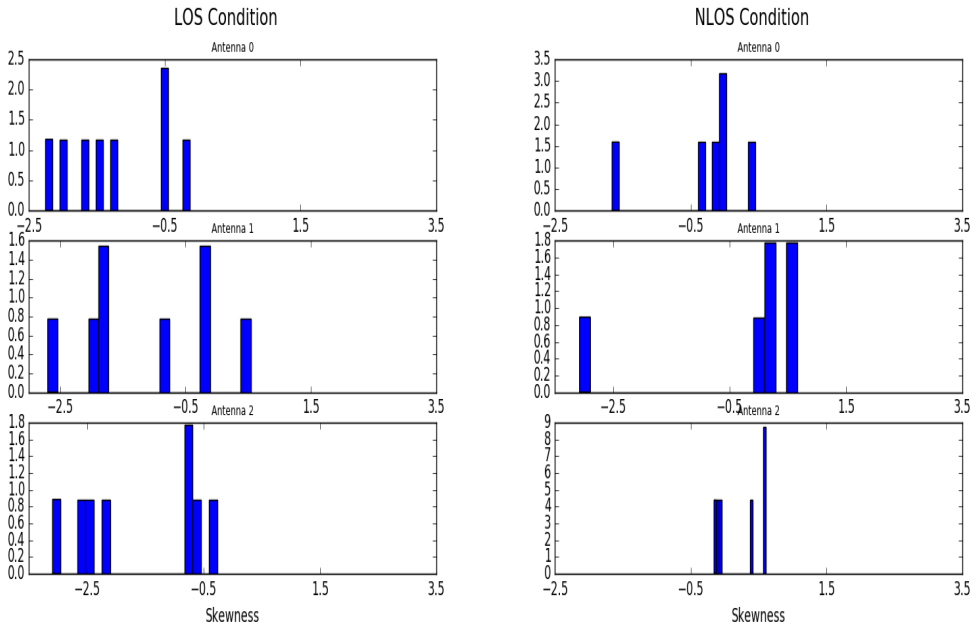


Fig. 4.10.: CSI histogram

We notice that the samples tend to be more symmetrical in the NLOS condition, we will use the skewness metric candidate to further analyze if the candidate is reliable for LOS identification, we display the histogram of the skewness with mean values for each antenna.



Skewness	Antenna 0	Antenna 1	Antenna 2
LOS	-1.2	-1.11	-1.57
NLOS	-0.29	-0.24	0.21

Fig. 4.11.: Skewness histogram and mean values

The result is confirming the observed shape of the CSI amplitude samples, in the NLOS conditions we observe a more symmetrical histogram compared to the LOS condition.
Regarding the K-factor, a similar test performed (figure 4.12) showed that the values for LOS conditions tend to be lower than the ones of NLOS conditions which is counter intuitive regarding the meaning of the K-factor (the fraction of power contained in the LOS component with respect to the NLOS one).
We thus decide to utilize only the skewness for the CSI.

Using only the skewness for CSI we achieve a percentage of success of:

- 66% for LOS recognition.
- 79% for NLOS recognition.

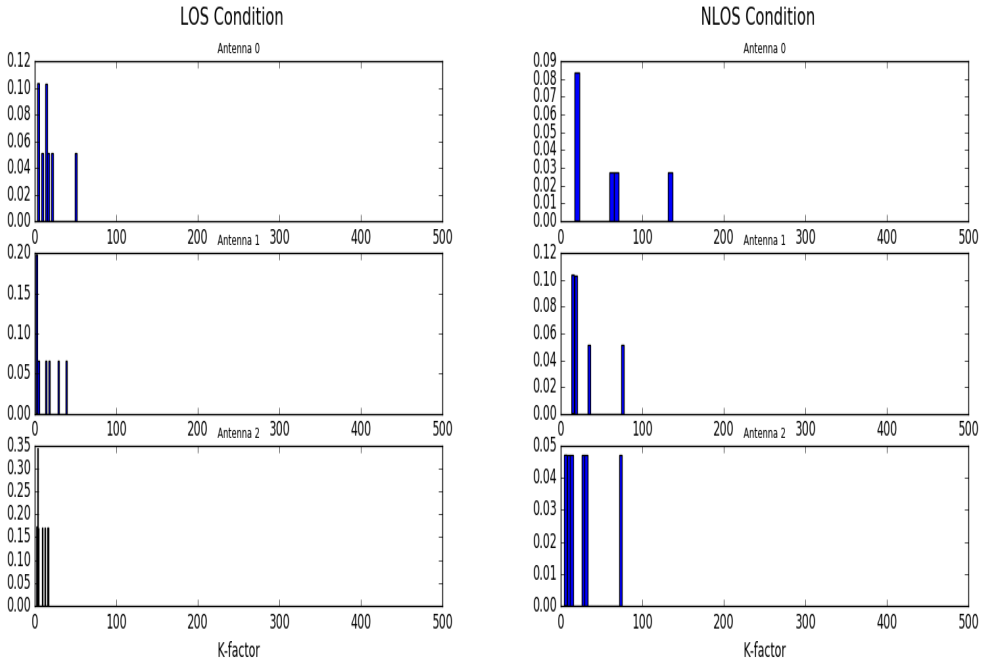


Fig. 4.12.: Values of K-factor for LOS and NLOS conditions

We see now how this percentage changes with the number of packets:

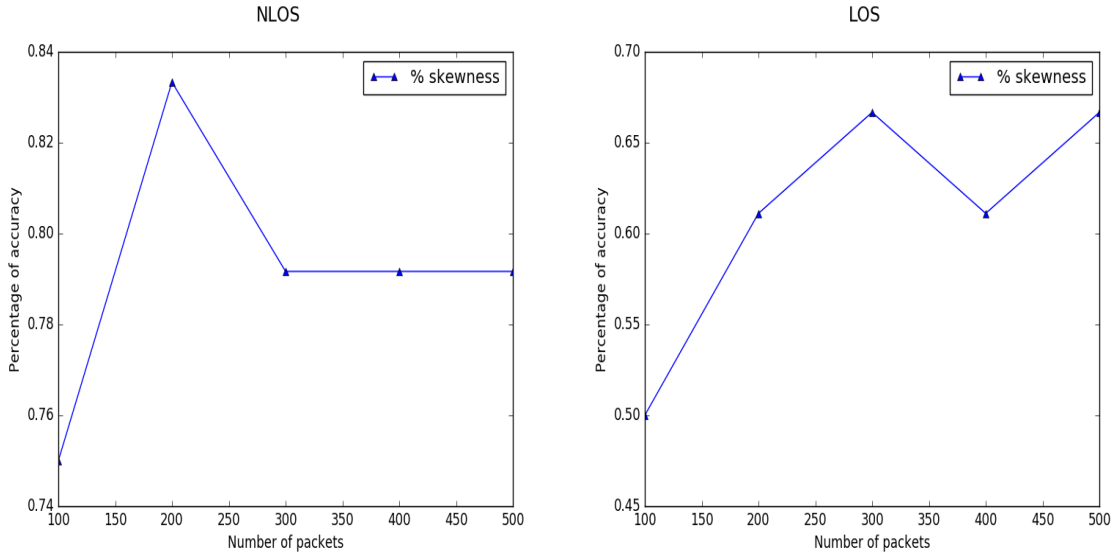


Fig. 4.13.: Percentage of success with number of packets

A number of 300 packets is enough to achieve the same percentage as for all packets. In our experiment it takes 3 seconds to monitor this number of packets.
We conclude that the CSI metric is achieving a better performance with respect to the percentage of success using the skewness but also regarding the time needed to get the result.

4.1 Recapitulation of LOS identification results

- We performed a RSSI histogram fitting with the state of the art distributions. Rice for line of sight communication and Rayleigh for non line of sight communication. We found that the fitting doesn't confirm the ground truth.

- We used two metrics from the LOS identification work [3] that determine the LOS from NLOS. The skewed shape of the RSSI histogram showed a symmetry in the NLOS RSSI histogram and an asymmetry in the LOS condition.

- We achieves a success rate of:

  - 63% for LOS identification and 66% for NLOS identification for the case of RSSI using the K-factor metric.

  - 66% for LOS identification and 70% for NLOS identification for the case of RSSI using the skewness metric.

  - 66% for LOS identification and 79% for NLOS identification for the case of CSI using the skewness metric.

The Fact that the LOS identification only achieves 66% while the ground truth states a LOS case proves that the multi-path transmission has a consequent impact on the RSSI. It introduces an amount of uncertainty with respect to the LOS identification. It further proves that the correlation of factors highly impacts the RSSI and introduces a fitting error with respect the state of the art models and the metrics used in [3] for the same purpose.

5 Conclusion

Traditionally, wireless protocol proposals have been often tested and validated using only analytical and simulation models. Experimentation is a mandatory step before possible deployment of new network protocols with real users. However, wireless experimentation is much more complex to set up and run than simulation. R2lab is a testbed for running network experiments that offers the possibility to evaluate the protocols. This work is offering guidelines for running trustful evaluation in the wireless environment of R2lab. We construct a radio map of the environment by finding and defining the effect of controllable and non controllable factors over the RSSI metric. We then fit the transmission inside the lab with a state of the art model for RSSI prediction. We also propose a candidate metric to recognize LOS communication with a success rate of 66%.

5.1 Future Work

The same methodology can be extended and used with other metrics such as packet loss, throughput etc. In fact we generated traffic using the udp transport protocol, one can generate a continuous flux of data over tcp and assess changes on the metric.

List of Figures

1.1 pros and cons of wireless environment 2
1.2 View of R2lab 3

2.1 Experiment design 5

3.1 Nodes evolved in the experiment 9
3.2 Interaction plot of Factors 10
3.3 Heatmap of R2lab in 2GHz and 5GHz 11
3.4 Caption for LOF 12
3.5 Heatmap of noise in 2GHz and 5GHz 12
3.6 Possible relative antenna positions for direct neighbors 13
3.7 Nodes involved 13
3.8 Histograms of received signal strength for each antenna 14
3.9 Interactive heatmaps 15
3.10 Received signal strength over time 15
3.11 Received signal strength over time 16
3.12 Nodes involved in the experiment 18
3.13 Signal strength with distance along with models 18

4.1 Illustration of the K-S test. Blue line is the ecdf of the data, red line is the fitting cdf, black arrow is the K-S statistic 21
4.2 Rice fitting of received signal strength histogram in LOS case 22
4.3 Rayleigh fitting of received signal strength histogram in LOS case . . . 23
4.4 Rayleigh fitting of received signal strength histogram in NLOS case . . 23
4.5 The design and test set of nodes 25
4.6 Histograms and mean values for Skewness and K-factor in LOS 25
4.7 Histograms and mean values for Skewness and K-factor in NLOS 26
4.8 LOS and NLOS test 27
4.9 Impact of number of packets on the success rate 28
4.10 CSI histogram 29
4.11 Skewness histogram and mean values 30
4.12 Values of K-factor for LOS and NLOS conditions 31
4.13 Percentage of success with number of packets 31

Bibliography

[1] Ramon Dos Reis Fontes, Mohamed Mahfoudi, Walid Dabbous, Thierry Turletti and Christian Rothenberg. *How far can we go? Towards Realistic Software-Defined Wireless Networking Experiments.* The Computer Journal (Oxford), Oxford, 2017. <hal-01480973>

[2] Cristian Tala, Luciano Ahumada, Diego Dujovne, Shafqat-Ur Rehman, Thierry Turletti and Walid Dabbous. *Guidelines for the accurate design of empirical studies in wireless networks.* TRIDENTCOM, volume 90 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, page 208-222. Springer, (2011)

[3] Zimu Zhou, Zheng Yang, Chenshu Wu, Wei Sun and Yunhao Liu . *LiFi: Line-Of-Sight Identification with WiFi.* CSE, Hong Kong University of Science and Technology. School of Software and TNList, Tsinghua University.

[4] https://r2lab.inria.fr/index.md

[5] Santosh Choudhary ,Dinesh Kumar Dhaka. *Path loss Prediction Models for Wireless Communication Channels and its Comparative Analysis.* International Journal of Engineering, Management & Sciences(IJEMS)ISSN-2348–3733,Volume-2,Issue-3, March 2015.

[6] Andrea Goldsmith . *WIRELESS COMMUNICATIONS.* Stanford University.

Appendices

Code to extract RSSI

A

A.1 Python3 script to automate the experiment

```
#!/usr/bin/env python3

import os

from argparse import ArgumentParser

from asynciojobs import Scheduler, Sequence, PrintJob

from apssh import SshNode, LocalNode, SshJob
from apssh import Run, RunScript, Pull
from apssh import TimeColonFormatter


##########
gateway_hostname  = 'faraday.inria.fr'
gateway_username  = 'inria_yassir'

# a fixed amount of time that we wait for once all the nodes
# have their wireless interface configured
settle_delay      = 10
# antenna mask for each node, three values are authorized:
   1, 3, 7
antenna_mask      = 7
# PHY rate used for each node, e.g. 1, 6, 54...
phy_rate          = 6
# Channel frequency used for each node
channel_frequency = 2412
# Tx Power for each node, for Atheros 5dBm (i.e. 500) to 14
   dBm (i.e. 1400)
tx          = 1400
#
# ping parameters
```

```
#
ping_timeout       = 1 #1205
ping_size          = 64
ping_interval      = 0.08  #0.008
ping_number        = 100 #100

parser = ArgumentParser()
parser.add_argument("-s", "--slice", default=
    gateway_username,
                    help="specify an alternate slicename,
                        default={}"
                          .format(gateway_username))
parser.add_argument("-l", "--load-images", default=False,
    action='store_true',
                    help = "enable to load the default image
                        on nodes before the exp")
parser.add_argument("-w", "--wifi-driver", default='ath9k',
                    choices = ['iwlwifi', 'ath9k'],
                    help="specify which driver to use")
parser.add_argument("-m", "--max", default=5, type=int,
                    help="will run on all nodes between 1
                        and this number")

parser.add_argument("-p", "--parallel", default=None,type=
    int,
                    help="""run in parallel, with this value
                         as the
                    limit to the number of simultaneous
                        pings - -p 0 means no limit""")
parser.add_argument("-a", "--antenna-mask", default=
    antenna_mask,choices = ['1','3','7'],
                    help="specify antenna mask for each node
                        - default={}".format(antenna_mask))
parser.add_argument("-r", "--phy-rate", default=phy_rate,
                    help="specify PHY rate - default={}".
                        format(phy_rate))
parser.add_argument("-f", "--channel-frequency", default=
    channel_frequency,
                    help="specify the channel frequency for
                        each node - default={}".format(
                        channel_frequency))
parser.add_argument("-T", "--tx-power", default=tx,
```

```
                    help="specify␣Tx␣power␣-␣default={}".
                        format(tx))
parser.add_argument("-t", "--ping-timeout", default=
    ping_timeout,
                    help="specify␣timeout␣for␣each␣
                        individual␣ping␣-␣default={}".format(
                        ping_timeout))
parser.add_argument("-i", "--ping-interval", default=
    ping_interval,
                    help="specify␣time␣interval␣between␣ping
                        ␣-␣default={}".format(ping_interval))
parser.add_argument("-S", "--ping-size", default=ping_size,
                    help="specify␣packet␣size␣for␣each␣
                        individual␣ping␣-␣default={}".format(
                        ping_size))
parser.add_argument("-N", "--ping-number", default=
    ping_number,
                    help="specify␣number␣of␣ping␣to␣send␣-␣
                        default={}".format(ping_number))

parser.add_argument("-n", "--dry-run", default=False, action
    ='store_true',
                    help="do␣not␣run␣anything,␣just␣print␣
                        out␣scheduler,␣and␣generate␣.dot␣file
                        ")
parser.add_argument("-v", "--verbose-ssh", default=False,
    action='store_true',
                    help="run␣ssh␣in␣verbose␣mode")
parser.add_argument("-d", "--debug", default=False, action='
    store_true',
                    help="run␣jobs␣and␣engine␣in␣verbose␣
                        mode")
parser.add_argument("-nu", "--number")

args = parser.parse_args()

max = args.max
gateway_username = args.slice
verbose_ssh = args.verbose_ssh
verbose_jobs = args.debug
ping_timeout = args.ping_timeout
ping_interval = args.ping_interval
```

```
ping_size = args.ping_size
ping_number = args.ping_number
wireless_driver   = args.wifi_driver
antenna_mask = args.antenna_mask
phy_rate = args.phy_rate
channel_frequency = args.channel_frequency
tx_power = args.tx_power

# convenience
def fitname(id):
    return "fit{:02d}".format(id)


###
# create the logs dirctory base don input parameters
dirlogs = "trace-freq{}-T{}-r{}-a{}-t{}-i{}-S{}-N{}-{}".
   format(channel_frequency,tx_power,phy_rate,antenna_mask,
   ping_timeout,ping_interval,ping_size, ping_number,args.
   number) #,args.number)

print("Creating log directory: "+dirlogs)
os.makedirs(dirlogs, exist_ok=True)

### the list of (integers) that hold node numbers, starting
   at 1
# of course it would make sense to come up with a less
   rustic way of
# selecting target nodes
node_ids = range(1, max+1)

########## the nodes involved
faraday = SshNode(hostname = gateway_hostname, username =
   gateway_username,
formatter=TimeColonFormatter(), verbose = verbose_ssh)

# this is a python dictionary that allows to retrieve a node
    object
# from an id
node_index = {
    id: SshNode(gateway = faraday,
                hostname = fitname(id),
                username = "root",
```

```
                formatter=TimeColonFormatter(),
                verbose = verbose_ssh)
    for id in node_ids
}

########## the global scheduler
scheduler = Scheduler(verbose = verbose_jobs)

##########
check_lease = SshJob(
    scheduler = scheduler,
    node = faraday,
    verbose = verbose_jobs,
    critical = True,
    command = Run("rhubarbe␣leases␣--check"),
)

########## load images if requested

green_light = check_lease

if args.load_images:
    negated_node_ids = [ "~{}".format(id) for id in node_ids
        ]
    # replace green_light in this case
    green_light = SshJob(
        node = faraday,
        required = check_lease,
        critical = True,
        scheduler = scheduler,
        verbose = verbose_jobs,
        commands = [
            Run("rhubarbe", "off", "-a", *negated_node_ids),
            Run("rhubarbe", "load", "-i", "ubuntu", *
                node_ids), #5ghz image: ubuntu-ath9k-ok .
                ndz
            Run("rhubarbe", "wait", *node_ids)
        ]
    )


init_wireless_jobs = [
```

```
    SshJob(
        scheduler = scheduler,
        required = green_light,
        node = node,
        verbose = verbose_jobs,
        label = "init {}".format(id),
        command =RunScript(
            "node-utilities.sh", "init-ad-hoc-network",
            wireless_driver, "foobar", channel_frequency,
               phy_rate, antenna_mask, tx_power)
        )
    for id, node in node_index.items() ]


######### then run tcpdump on fit nodes, this job never ends
    ...
run_tcpdump = [
    SshJob(
        scheduler = scheduler,
        node = node,
        required = init_wireless_jobs,
        label = "run tcpdump on fit nodes",
        verbose = verbose_jobs,
        commands = [
            Run("echo run tcpdump on fit{:02d}".format(i)),
#            Run("tcpdump -U -i moni0 -v icmp -y
    ieee802_11_radio -w /tmp/fit{}.pcap".format(i))
            Run("tcpdump -U -i moni0 -y ieee802_11_radio -w
               /tmp/fit{}.pcap".format(i))
        ]
    )
    for i, node in node_index.items()
]


########## let the wireless network settle
settle_wireless_job = PrintJob(
    "Let the wireless network settle",
    sleep = settle_delay,
    scheduler = scheduler,
    required = init_wireless_jobs,
    label = "settling")
```

```
##########
# create all the ping jobs, i.e. max*(max-1)/2
# this again is a python list comprehension
# see the 2 for instructions at the bottom
#
# notice that these SshJob instances are not yet added
# to the scheduler, we will add them later on
# depending on the sequential/parallel strategy



pings = []

for i, nodei in node_index.items():
 nodes = list(range(1,38))
 if i==37:
  j=1
 else:
  j=i+1


 opn = [SshJob(
        node = nodei,
        required = settle_wireless_job,
        commands = [
            Run("iw dev atheros survey dump | grep noise | 
                sed -n 1p >> /tmp/noise.txt") #The noise for
                the 2412 band, when using another frequency
                band, you should change the line picked by
                sed accordingly
        ]
        )
    for i,nodei in node_index.items() ]                #{k:
        node_index[k] for k in [i,j]}.items()

 for o in opn:
    pings.append(o)

 pings.append( SshJob(
        node = nodei,
```

```
            required = settle_wireless_job,
            label = "ping␣{}␣->␣{}".format(i, j),
            verbose = verbose_jobs,
            commands = [
                #Run("sleep 1"),
                Run("echo␣{}␣'->'␣{}".format(i, j)),
                RunScript("node-utilities.sh", "my-ping",
                          "10.0.0.{}".format(j), ping_timeout,
                             ping_interval,
                          ping_size, ping_number,
                          ">", "PING-{:02d}-{:02d}".format(i, j)
                             ),
                Pull(remotepaths = "PING-{:02d}-{:02d}".format(i
                    , j), localpath="./"+dirlogs),

            ]))


# retrieve all pcap files from fit nodes
retrieve_tcpdump = [
    SshJob(
            scheduler = scheduler,
            node = nodei,
            required = pings,
            label = "retrieve␣pcap␣trace␣from␣fit{:02d}".format(
                i),
            verbose = verbose_jobs,
            commands = [
                Run("sleep␣2;pkill␣tcpdump;␣sleep␣1"),
                RunScript("node-utilities.sh", "process-pcap", i
                    ),
                Run("echo␣retrieve␣pcap␣trace␣from␣fit{:02d}".
                    format(i)),
                Run("mv␣/tmp/noise.txt␣/tmp/noise{}.txt".format(
                    i)),
                Pull(remotepaths ="/tmp/noise{}.txt".format(i),
                    localpath="./"+dirlogs),
                Pull(remotepaths = "/tmp/fit{}.pcap".format(i),
                    localpath="./"+dirlogs),
                Run("rm␣/tmp/noise{}.txt".format(i)),
                Run("echo␣retrieve␣result{}.txt␣file".format(i))
                    ,
```

```
                Pull(remotepaths = "/tmp/result-{}.txt".format(i
                    ), localpath="./"+dirlogs),
                Run("rm /tmp/result-{}.txt".format(i))

            ]
        )
        for i, nodei in node_index.items()
]



if args.parallel is None:
    # with the sequential strategy, we just need to
    # create a Sequence out of the list of pings
    # Sequence will add the required relationships
    scheduler.add(Sequence(*pings, scheduler=scheduler))
    # for running sequentially we impose no limit on the
       scheduler
    # that will be limitied anyways by the very structure
    # of the required graph
    jobs_window = None
else:
    # with the parallel strategy
    # we just need to insert all the ping jobs
    # as each already has its required OK
    scheduler.update(pings)
    # this time the value in args.parallel is the one
    # to use as the jobs_limit; if 0 then inch'allah
    jobs_window = args.parallel


# Finally - i.e. when traces are retrieved from all nodes
# we can resume postprocessing of traces on the
   corresponding directory
SshJob(
    node = LocalNode(),
    scheduler = scheduler,
    required = retrieve_tcpdump,
    verbose = verbose_jobs,
    commands = [
        Run("echo Run post-process.py on {}".format(dirlogs)
            ),
```

```
            Run("cd {};".format(dirlogs), "python3 ../post-
                process.py -m {} ".format(max),
                 " -a {}".format(antenna_mask)),
        ]
)

#
# dry-run mode
# show the scheduler using list(details=True)
# also generate a .dot file, and attempt to
# transform it into a .png - should work if graphviz is
    installed
# but don't run anything of course
#
if args.dry_run:
    print("==================== COMPLETE SCHEDULER")
    # -n + -v = max details
    scheduler.list(details=verbose_jobs)
    suffix = "par" if args.parallel else "seq"
    if args.load_images:
        suffix += "-load"
    filename = "heatmap-{}-{}".format(suffix, args.max)
    print("Creating dot file: {filename}.dot".format(
        filename=filename))
    scheduler.export_as_dotfile(filename+".dot")
    # try to run dot
    command = "dot -Tpng -o {filename}.png {filename}.dot".
        format(filename=filename)
    print("Trying to run dot to create {filename}.png".
        format(filename=filename))
    retcod = os.system(command)
    if retcod == 0:
        print("{filename}.png OK".format(filename=filename))
    else:
        print("Could not create {filename}.png - do you have
            graphviz installed ?"
                .format(filename=filename))
    # in dry-run mode we are done
    exit(0)

ok = scheduler.orchestrate()    #jobs_window=jobs_window
# give details if it failed
```

```
ok or scheduler.debrief()

# return something useful to your OS
exit(0 if ok else 1)
```

A.2 Bash script to interact with wireless card

```
#!/bin/bash

####################
# This is our own brewed script for setting up a wifi
    network
# it run on the remote machine - either sender or receiver
# and is in charge of initializing a small ad-hoc network
#
# Thanks to the RunString class, we can just define this as
# a python string, and pass it arguments from python
    variables
#

# we expect the following arguments
# * wireless driver name (iwlwifi or ath9k)
# * the wifi network name to join
# * the wifi frequency to use

function init-ad-hoc-network (){
    driver=$1; shift
    netname=$1; shift
    freq=$1;    shift
    phyrate=$1; shift
    antmask=$1; shift
    txpower=$1; shift
  #  ipipaddr_mask=$1; shift

    # load the r2lab utilities - code can be found here:
    # https://github.com/parmentelat/r2lab/blob/master/infra
       /user-env/nodes.sh
    source /root/r2lab/infra/user-env/nodes.sh

    # make sure to use the latest code on the node
    git-pull-r2lab
    iw reg set AW
   # turn-off-wireless

    ipaddr_mask=10.0.0.$(r2lab-ip)/24
```

```
  echo loading module $driver



#  modprobe $driver
#  iw reg set AW
  iw dev moni0 del

  # some time for udev to trigger its rules
  sleep 1

  # install tshark on the node for the post-processing
     step
  apt-get install tshark
  ifname=atheros #$(wait-for-interface-on-driver $driver)
  phyname=`iw $ifname info|grep wiphy |awk '{print "phy"$2
     }'`

  echo configuring interface $ifname
  # make sure to wipe down everything first so we can run
     again and again
  ip address flush dev $ifname
  echo "ip␣link␣set␣$ifname␣down"
  ip link set $ifname down
  # configure wireless
  echo "iw␣phy␣$phyname␣set␣antenna␣$antmask"
  iw phy $phyname set antenna $antmask     # phy1 <> phy0
  iw dev $ifname set type ibss
  # set the Tx power Atheros'range is between 5dbm (500)
     and 14dBm (1400)
  sleep 1
  ip link set $ifname up
  # set to ad-hoc mode and set the right PHY rate
  echo "iw␣dev␣$ifname␣ibss␣join␣$netname␣$freq"
  iw dev $ifname ibss join $netname $freq
  sleep 1
  echo "iw␣dev␣$ifname␣set␣txpower␣fixed␣$txpower"
  iw dev $ifname set txpower fixed $txpower
  echo "POWER!!"
  iwconfig atheros | grep Tx-Power
  ip address add $ipaddr_mask dev $ifname
  if test $freq -eq 2412
```

```
    then
        iw dev $ifname set bitrates legacy-2.4 $phyrate
    else
        iw dev $ifname set bitrates legacy-5 $phyrate
    fi
    echo "iw dev $ifname set bitrates legacy-2.4 $phyrate"

    # set the wireless interface in monitor mode
    iw phy $phyname interface add moni0 type monitor
    ip link set moni0 up

    # then, run tcpdump with the right parameters

#    tcpdump -U -W 2 -i moni0 -y ieee802_11_radio -w "/tmp/"
   $(hostname)".pcap"


    ### addition - would be cool to come up with something
       along these lines that
    # works on both cards
    # a recipe from Naoufal for Intel
    # modprobe iwlwifi
    # iwconfig wlan2 mode ad-hoc
    # ip addr add 10.0.0.41/16 dev wlan2
    # ip link set wlan2 up
    # iwconfig wlan2 essid mesh channel 1

}

function my-ping (){
    dest=$1; shift
    ptimeout=$1; shift
    pint=$1; shift
    psize=$1; shift
    pnumber=$1; shift

    echo "ping -W $ptimeout -c $pnumber -i $pint -s $psize -
       q $dest >& /tmp/ping.txt"
    ping -w $ptimeout -c $pnumber -i $pint -s $psize -q
       $dest >& /tmp/ping.txt
    result=$(grep "%" /tmp/ping.txt)
    echo "$(hostname) -> $dest: ${result}"
```

```
    return 0
}


function process-pcap (){
    i=$1; shift
    array="1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
       20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
       "
    for j in `echo ${array[@]/$i}`
    do
      tshark -r /tmp/fit"$i".pcap  -Y "ip.src==10.0.0.$j &&
         icmp"  -Tfields -e "ip.src" -e "ip.dst" -e "
         radiotap.dbm_antsignal" >> /tmp/result"-$i".txt
    done
    node=$1; shift
    tshark -2 -r /tmp/fit"$node".pcap  -R "ip.dst==10.0.0.
       $node && icmp"  -Tfields -e "ip.src" -e "ip.dst" -e "
       radiotap.dbm_antsignal" -e "radiotap.dbm_antnoise" >
       /tmp/result"-$node".txt
    echo "Run tshark post-processing on node fit$node"
    return 0
}



##########################################
# just a wrapper so we can call the individual functions. so
    e.g.
# node-utilities.sh tracable-ping 10.0.0.2 20
# results in calling tracable-ping 10.0.0.2 20

"$@"
```

A.3 Python3 script for post processing

```
#!/usr/bin/env python3

# Compute average RSSI values for each node and fill values
    when missing with either RSSI_MIN or RSSI_MAX
# - input RSSI files "result-X.txt" obtained through tshark
    for each FIT "receiving" node contain RSSI values
    corresponding
# to ICMP ping packets sent by other FIT nodes. The number
    of RSSI values for each ping depends on the number of
    antennas
# used.
# - output RSSI files "rssi-X.txt" contain the average RSSI
    values received at node X.
#
# TT 20/3/17


import os

RSSI_MAX = 0
RSSI_MIN = -100


# convenience
def fitname(id):
    return "fit{:02d}".format(id)

def mask_to_nb(mask):
    switcher = {
        1: 1,
        3: 2,
        7: 3,
    }
    return switcher.get(mask)

def save_RSSI(file_RSSI, node_max, RSSI):
    """
    write to file the overall RSSI values
    """
```

```
    [[file_RSSI.write(RSSI[s][r]) for r in range(1, node_max
       )] for s in range(1, node_max)]
    return 0

def store_missing_rssi(file_out, Nant, node, sender,
   receiver, RSSI):
    """

       write to the output file the missing  RSSI value for
       the couple sender, receiver
                                                  i.e.,
       RSSI_MAX if the node itself is sending or else
       RSSI_MIN
    """
    if sender == node:
        value = RSSI_MAX
    else:
        value = RSSI_MIN
    output = "10.0.0.{:02d}\t10.0.0.{:02d}\t".format(sender,
        receiver)
    for i in range(Nant+1):
        output += "{}\t".format(value)
    output += "\n"
    file_out.write(output)
    RSSI[sender][receiver] = output
    return 0



def store_rssi(file_out, Nant, sender, receiver, sum_rssi,
   Nval, RSSI):
    """
    write to the output file constant RSSI values for the
       couple sender, receiver
    """
    output = "10.0.0.{:02d}\t10.0.0.{:02d}\t".format(sender,
        receiver)
    for i in range(Nant+1):
        output += "{0:.2f}\t".format(sum_rssi[i]/Nval)
    output += "\n"
    file_out.write(output)
    RSSI[sender][receiver] = output
```

```
    return 0


def process(file_in, file_out, node, node_max, Nant, RSSI):
    """
    expects a FIT node number and the number of antennas and
        postprocess data
    lines of file_in for receiver Y are in the following
       format:
    10.0.0.X    10.0.0.Y        A,B,C,D
    where X is for sender, Y for receiver, A..D integer
       values for RSSI,
    if Nant==1, only A and B, if Nant==3, A,B,C,D RSSI
       values are included
    """

    cur_sender = 0
    expected = 1

    for line in file_in:
        # Read RSSI values one line at a time
        sender = int(line.split()[0].split('.')[3])
        #receiver = int(line.split()[1].split('.')[3])
        receiver = node
        rssi  = [int(line.split()[2].split(',')[i]) for i in
            range(Nant+1)]

        if sender == cur_sender:
            # Yet another RSSI value for cur_sender to
               process
            sum_rssi = [sum_rssi[i]+rssi[i] for i in range(
               Nant+1)]
            Nval += 1
        else:
            # This sender is new
            if cur_sender:
                # Store RSSI computed for cur_sender as no
                   more data for it
                store_rssi(file_out, Nant, cur_sender,
                   receiver, sum_rssi, Nval, RSSI)
                expected += 1
```

```
            while expected < sender:
                # handle the case when data is missing for
                    some senders
                store_missing_rssi(file_out, Nant, node,
                    expected, receiver, RSSI)
                expected += 1
            # At this point, sender is the one expected and
                it is the first RSSI value to store
            sum_rssi = rssi
            Nval = 1
            cur_sender = sender
    # At this point there is no more data in file
    # we need to store RSSI computed for the current sender
    # and handle the case when the current sender is not the
        last sender in the list
    if cur_sender:
        # Store RSSI computed for cur_sender as no more data
            for it, receiver==node
        store_rssi(file_out, Nant, cur_sender, receiver,
            sum_rssi, Nval, RSSI)
        expected += 1
    while expected < node_max:
        # handle the case when data is missing for some
            senders
        store_missing_rssi(file_out, Nant, node, expected,
            node, RSSI)
        expected += 1
    # we are done

    return 0



def main():
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument("-m", "--max", default=37, type=int,
                        help="node␣number␣vary␣between␣1␣and
                            ␣this␣number")
    parser.add_argument("-a", "--ant-mask", default=7,type=
        int,choices = [1,3,7],
```

```
                                help="specify the mask of antennas
                                    for each node - default is 1")

    parser.add_argument("-v", "--verbose", action='
        store_true', default=False)
    parser.add_argument("-d", "--debug", action='store_true'
        , default=False)

    args = parser.parse_args()

    node_max = args.max+1
    ant_mask = args.ant_mask
    Nant = mask_to_nb(ant_mask)

    pos =
        [[0,0],[1,5],[1,4],[1,3],[1,2],[1,1],[2,5],[2,4],[2,3],[2,2],[2,1],[3,5]

            [3,2],[3,1],[4,5],[4,3],[4,2],[5,5],[5,4],[5,3],[5,2],[6,5],[6,3],[6

            [7,3],[7,2],[7,1],[8,5],[8,4],[8,3],[8,2],[8,1],[9,2],[9,1]]

    RSSI = [["" for sender in range(1, node_max+1)] for
        receiver in range(1, node_max+1)]
    file_RSSI = open("RSSI.txt", "w")

    for node in range(1, node_max):
        filename_in = "result-{}.txt".format(node)
        filename_out = "rssi-{}.txt".format(node)
        file_in = open(filename_in, "r")
        file_out = open(filename_out, "w")
        process(file_in, file_out, node, node_max, Nant,
            RSSI)
        file_in.close()
        file_out.close()
    save_RSSI(file_RSSI, node_max, RSSI)
    file_RSSI.close()

    exit(0)

if __name__ == "__main__":main()
```